Optical Character Recognition of Japanese Kanji

ISYE 6740 - Spring 2022 - Project Final Report

Bharathrham Kodungudi
GT ID: 903245870
bharathrham@gatech.edu

Mohnish Shah
GT ID: 903657060
shahmohnish@gatech.edu

1 Introduction

The Japanese language contains three character sets. Two syllabaries: Hiragana and Katakana; one logography: Kanji. Kanji came to Japan from China around the 6^{th} century CE, along with Buddhism, music and silk textiles. The Chinese logography is built such that each character represents a concept or idea, and can be made up of smaller portions called radicals. For example: 日, meaning day; and 月 meaning month or moon are both used in 明 meaning tomorrow. Radicals often inform on the meaning or pronunciation of a character they show up in, although occasionally they do not.

These Chinese characters were initially adopted to both represent a sound as well as the original meaning. For example 有 (to exist) also represented the sound 'A'. Furthermore, Kanji tend to have multiple pronunciations depending on context and surrounding characters, which further confuses readers. Over generations of writers and poets modified, simplified, and eventually standardized the characters into the first Japanese syllabary: Hiragana. In Hiragana, the sound 'A' is represented by あ, which when analyzed shows its ties to its roots in the original 有. Later a second syllabary, Katakana, was invented using a similar but different approach to simplifying Chinese logographs to represent sounds alone.

Since the invention of the syllabaries, Hiragana, Katakana and Kanji are all used in conjunction. Hiragana to represent spoken sounds, Kanji to represent concepts in a more compressed form. This complication of the language into multiple sets of characters with different design and different rules presents an interesting challenge to the task of optical character recognition.

2 Problem Statement

The goal of this project is less to build a powerful model, but more to understand how various models and methods perform the task of OCR. Since Hiragana and Kanji are always side by side in a sentence, it made the most sense to deal with these together using two different strategies. The first strategy was to train a single model to classify both the syllabary and the logography at once. The second strategy was to create an ensemble of models, first to train a model to classify whether a character was Hiragana or Katakana, then to train two different models, one for Hiragana and one for Kanji to classify each more accurately. Finally, we will distill our learnings to achieve a single model with the highest accuracy rate.

3 Data Source

3.1 ETL Database

The data source we used is the ETL Character Database, found here: http://etlcdb.db.aist.go.jp/. It records handwritten English characters, Arabic numerals, Japanese Hiragana, Katakana and Kanji.

There are nine datasets in the ETL database, each one from a different source and containing different data. ETL1 contains Arabic numerals, capital English letters and Japanese Katakana characters, written by hundreds of volunteers. ETL2 contains computer fonts. ETL3 contains English letters and symbols, ETL 4 contains Hiragana. ETL 5 contains Katakana. ETL8 and ETL9 contain over 3000 kanji characters, handwritten several times by a few volunteers.

The diversity in the data allowed us to build OCR models with ranging complexity. For this project, we chose to use ETL4, ETL8 and ETL9, which gave us all the Hiragana characters and many Kanji characters.

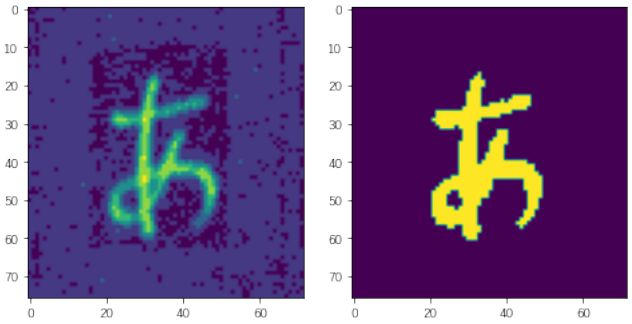

Figure 1: Example of denoising and binalizing a character image.

3.2 Denoising and Pre-processing

Figure 9, which can be found on the last page, shows an example image contained in ETL4. The image shown consists of 40 rows of 50 characters in each row. Data pre-processing included running a provided script to extract the images from a compressed file. Once we had all our images, the next step was to extract each individual character and store its corresponding label, found in an accompanying text file.

Once the images were extracted and labelled, we realized that there was some further denoising necessary before we could run any models. Looking at the raw image, it is clear that some characters

are lighter than others. We do not want the strength of the pen or pencil stroke to impact our models. Finally, the background of the paper itself is not perfectly white and had some artifacting.

Some of the datasets had character images 63 pixels by 64 pixels, while some others consisted of 76 pixels by 72 pixels. Not only were these images inconsistent and therefore impossible to include together in the same model, but they were too large and would needlessly complicate the models trained. For this reason, all images were downsampled to 32x32 prior to training any models.

4 Methodology

4.1 Principal Component Analysis

We performed PCA on our data to build our random forest, SVM, and KNN models. We have named the principal components of the Hiragana characters and Kanji characters 'Eigengana'. Understanding the patterns that arise in these Eigengana is fundamental to understanding how the machine learning models that use them perform. The basis of this analysis is the concept of directions of maximal variance in the data.

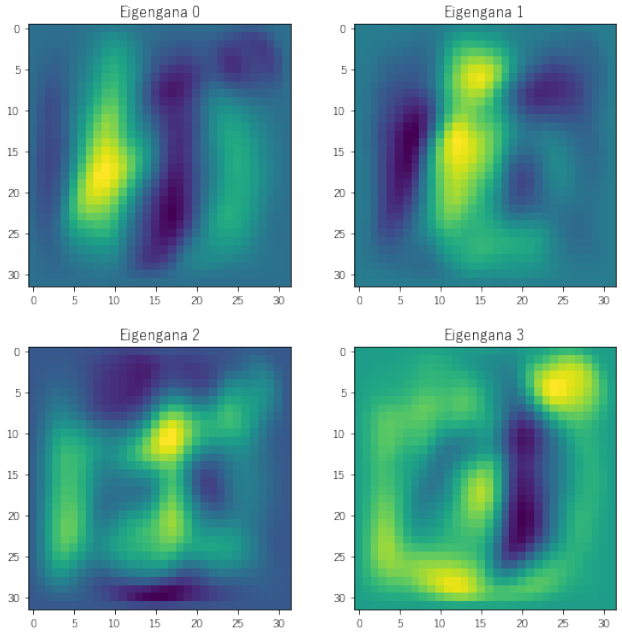

Figure 2: Top 4 Eigenvectors produced by PCA on Hiragana character images (Eigengana).

4.1.1 Eigengana One feature that is prominent in all Eigengana is the marker of a voiced character. The distinction between さ(SA) and ざ(ZA) is the voicing of the first sound. This is represented by two tick-marks to the top-right of the character, and is an option for most syllables to have. In Eigengana 0, we see a negative weight being given to the voicing marker, whereas in Eigengana 3 we see significant positive weight being given to the marker.

Other shapes that show themselves in the Eigengana are left-hand upstroke as can be seen in は、ほ、わ、ね、れ. Once again, we see this pattern show up in positive weight in Eigengana 2 and to an extent Eigengana 0, and in a negative weight in Eigengana 1.

Eigengana 2 looks remarkably like ば, but this is not entirely surprising as は、ば、ば、ま、ほ、ほ、ぼ are all distinct, but similar shaped characters which make up 7 of the 71 voiced and unvoiced hiragana. Clearly this pattern occurred often enough for it to be one of the dimensions of maximal variance and show up in the Eigengana.

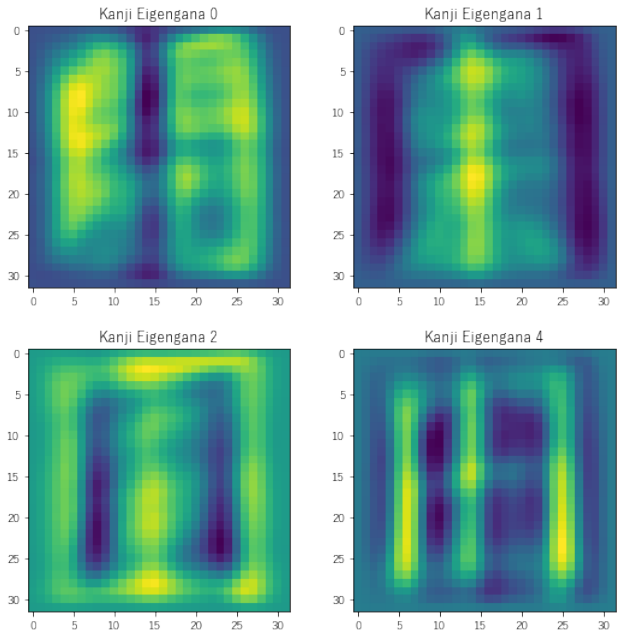

Figure 3: Top 4 Eigenvectors produced by PCA on Kanji character images (Kanji Eigengana).

4.1.2 Kanji Eigengana The Kanji Eigengana also exhibit interesting patterns that inform us on the patterns that are picked up by PCA. The first thing to be noted are the fundamental differences between the Eigengana and the Kanji Eigengana. Kanji in general exhibit more straight lines, right angles and geometric shapes, whereas hiragana are curvy, flow from stroke to stroke and are more variable in their shapes. Another distinction is the codified radicals that are present in Kanji, but not in Hiragana. All kanji are either simple enough to be a radical themselves, like 木 (tree) or 田 (rice paddy); or are comprised of multiple radicals, like 働(work) or 果(fruit).

Kanji Eigengana 0 clearly contains the *Gate* radical 門, which is present in several kanji like 間(space), 問(question), 関(relation), 聞(to hear), 開(to open). The left side of the Kanji Eigengana is a

bit blurry and ambiguous, and could broadly relate to having any distinct radical on the left, such as the *Hand* radical 扌being on the left side of 打(to strike).

Kanji Eigengana 1 is quite distinct from the others shown here, and looks remarkably like 束(bundle), drawing from the *Tree* radical at the bottom. Once again, however, this pattern shows up in many places such as 杲(clear) or 果(fruit). Another key characteristic that this Kanji Eigengana picks up on is a straight line down the middle. This attribute is also present in Kanji Eigengana 4, and the straight, central line is an attribute that shows up in many, many kanji. Much like we saw for the Eigengana, there is significant negative weight given to that same pattern.

Kanji Eigengana 2 has a large *Box* radical around the outside 冂, but this shape also captures the *Upside-down Box* radical 冖, and includes the characters 回(revolutions), 同(same), 国(country).

To determine the optimum number of Eigengana, we ran various models using different numbers of Eigengana and compared accuracy scores. We noticed that around 47 Eigengana, most models started seeing negligible improvement in accuracy.

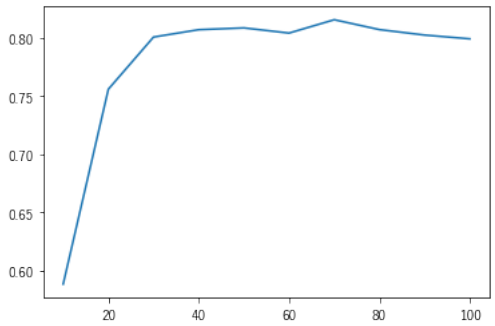

Figure 4: Number of Eigengana vs Accuracy for Random Forest

Figure 4 shows how the accuracy for a random forest increases as the number of Eigengana increase. There marginal improvement significantly decreases at approximately 30 Eigengana, but improvements continue until 40 Eigengana.

Repeating the same process for multiple machine learning models, we realized that at 47 Eigengana, on average, most models saw negligible improvements to accuracy.

We used 47 Eigengana to run the following three models:Random Forest, Support Vector Machine, K Nearest Neighbors
We ran each model 4 times. Once with just the hiragana data set, where we classified each individual hiragana character. Once with just the kanji data set, where we classified each kanji character. Once with a combined hiragana and kanji data set, where we classified each individual character. And lastly, with the combined data set, where we classified each character as either hiragana and kanji.

The reason for this fourth model was to see if we could improve our accuracy by first classifying a character as hiragana or kanji and then running the optimal model for hiragana/kanji to determine which character it was.
For each of the Random Forest, SVM, and KNN models we broke our data into splits of training and testing. We used 80% of the data for training and 20% of the data for testing. For each model, we split the data before doing any tuning. This way each model would be compared equally.

4.2 Random Forests

A random forest is a supervised classification model. For our random forest, we used bagging to create several random training samples. For each training sample, we created a decision tree. The Random Forest's outputted classification is the character that the majority of individual trees selected.

Table 1: Random Forest Accuracy

Model	Accuracy
Hiragana	82.32%
Kanji	82.80%
Combined	81.69%
Hiragana vs Kanji	90.65%

When looking at variable importance for all the data sets, an interesting pattern emerges. The variable importance matches nearly perfectly with the order of Eigengana. The most important variable is the first Eigengana. The second most important variable is the second Eigengana. And so on so forth. This follows intuition as PCA finds the Eigengana, in order, that maintain the most variability across the data. It is not surprising then that the top Eigengana are the most important variables in a random forest.

4.3 Support Vector Machines

When building our support vector machines, we tried various kernels, C values, and gammas.

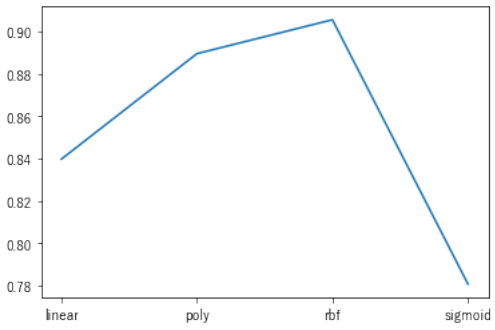

Figure 5: Combined Data Set SVM Accuracy across different Kernels

Figure 5 shows that for the combined data set, the RBF kernel had the highest accuracy score. This is before tuning either the C or gamma values. Because the best kernel is the RBF kernel, we can conclude that the characters are not linearly separable.

We then tuned the C-value and gamma parameters for the RBF SVM. We did this for the following models: hiragana only, kanji only, combined, Hiragana vs Kanji.

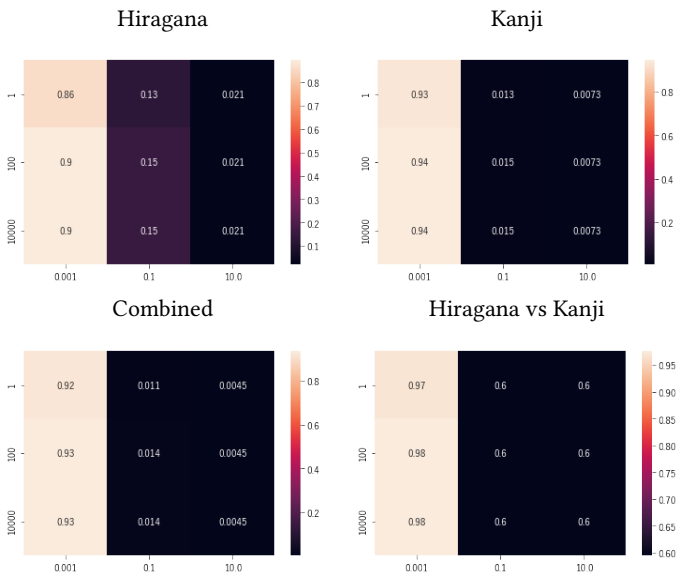

Figure 6: SVM accuracy scores across c and gamma values

The top left plot is in regards to the Hiragana only model. The top right is the Kanji only model. The bottom left is the combined model and the bottom right is the Hiragana vs Kanji model. For each plot, the x-axis axis is the gamma value and the y-axis is the c-value.

In each model, the accuracy improves as the gamma becomes smaller and the c-value becomes larger. The c-value determines the trade off between error and maximizing the margin of the decision boundaries. Gamma decides how much curvature exists in the decision boundaries. Higher gamma means more curvature and lower gamma means less curvature.

Gamma has a large impact on the SVM models. For each model, high gammas are associated with sub 1% accuracy. However at high gammas, every model crosses the 90% accuracy threshold. This is a huge improvement. The c-value has a lesser impact to the SVM's accuracy. However we are seeing that for each model, the higher c-values have higher accuracy. A higher c-value prevents over-fitting to the training data. Because high c-values are preferred, we know that our training data has a tendency to over-fit.

The final accuracy for each model are as follows.

Table 2: SVM Accuracy

Model	Accuracy
Hiragana	90%
Kanji	94%
Combined	93%
Hiragana vs Kanji	98%

4.4 KNN

The third model we tried was K-nearest-neighbors. For this model, the parameter we tuned was the k-number of nearest neighbors.

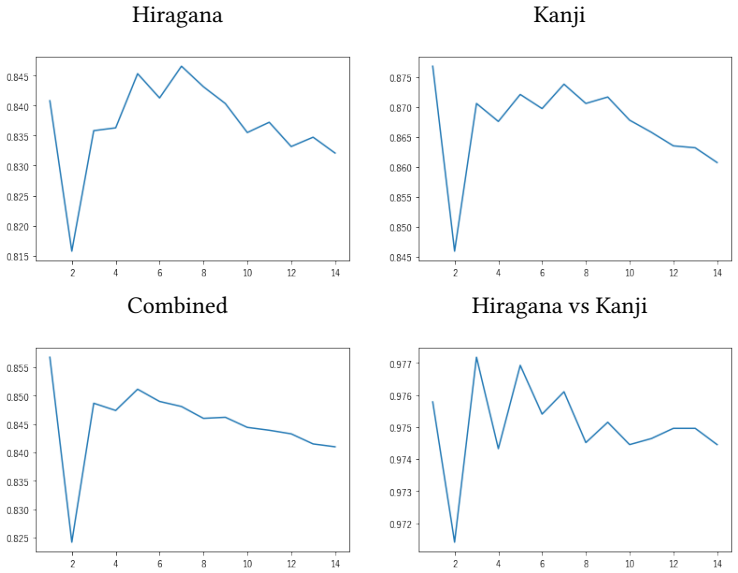

Figure 7: KNN accuracy scores across k nearest neighbors

For 3 of the 4 models, k = 1 results in the highest accuracy. However, k = 1 over-fits to the training data and should be disregarded. Therefore, for the Hiragana and Kanji models, k = 7 results in the best model. For the combined model, k = 5 is the best model. Finally, for the Hiragana vs Kanji model, k = 3 is the most accurate.

Table 3: KNN Accuracy

Model	Accuracy
Hiragana	85.48%
Kanji	87.69%
Combined	85.31%
Hiragana vs Kanji	97.22%

4.5 Neural Networks

Neural networks are the the most common tool used for OCR, so it was necessary for us to explore this to compare with the other models. Neural nets perform best with the raw data and not with PCA. The vast number of options and combinations when building a Neural Network made optimizing a challenge. Choosing an appropriate number of hidden layers, an appropriate loss function, an appropriate time to train and validate the models.

One of the questions initially posed for this project was understanding how neural networks learn, and whether we would see patterns of radicals in the initial weight vectors of the neural network. We already saw how these patterns might show themselves when we analyzed the Eigengana.

After thorough experimentation, we found the best loss function to be Categorical Crossentropy, using a ReLu activation function. We used a 20% split of the data for testing, and an additional 10% split for cross validation when training the neural networks through around 30 epochs.

4.5.1 Densely Connected Hidden Layers are the quintessential model of neural networks. Each pixel is connected to to every single node in the layer, and each connection carries an individual weight. The benefit of dense layers is that no preexisting structure is assumed for the data, and all learning happens through modifying the weights.

There are two ways to modify a dense neural network, by increasing the number of hidden layers or by increasing the number of nodes in a given layer, or of course, a combination of both. Below is a limited table of results from training densely connected neural networks of various sizes. In our expansive testing, we found that outside of a few rare cases, the testing accuracy score was not able to break past 90% for any configuration. This limitation seems intrinsic to dense neural networks in general, as many of these models reached 99% or 100% training accuracy and did not improve further in the validation or test sets in further epochs.

Table 4: Dense Neural Network Accuracy

Model	Hiragana	Kanji	Combined
Dense(16)	78.3	75.4	72.9
Dense(32)	84.8	81.9	79.5
Dense(64)	84.1	85.3	83.0
Dense(256)	87.8	89.1	86.6
Dense(512)	89.3	91.1	87.9
Dense(64,512)	85.4	83.4	82.4
Dense(512,64)	91.2	89.2	87.1
Dense(512,512)	90.7	89.6	88.8
Dense(512,512,64)	87.4	89.9	87.6
Dense(1024,512,256)	91.3	90.9	89.6

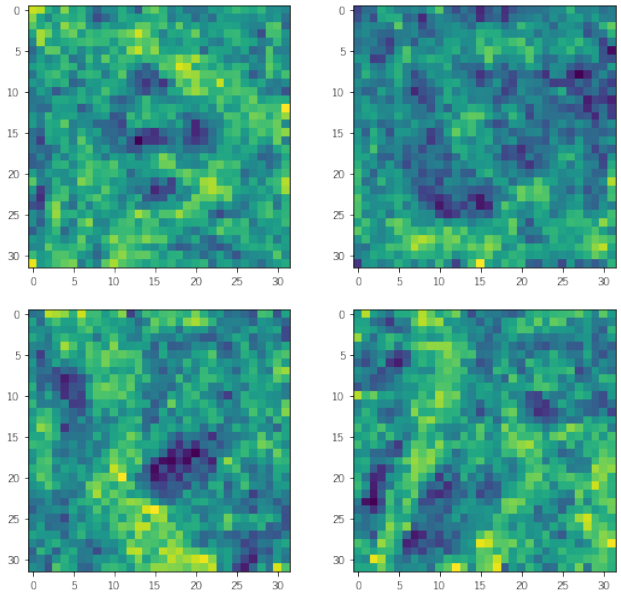

Figure 8: Four (of many) weight vectors produced by the first densely connected hidden layer.

The resulting weight vectors shown in Figure 8 are fascinating for their lack of patterns and structure. The weights shown are from the first layer of a densely connected neural network trained on only Kanji characters that achieved a 91% test accuracy rate, configured with 3 hidden layers of 128 nodes, 512 nodes, and 256 nodes respectively. After plotting and reviewing the weight vectors of many such models, we elected to only show these as the most patterned and interesting set of weight vectors with quite a high accuracy score. We expected to see something indicating the overall structure of the characters, especially for the Kanji which have very rigid components, straight lines, and right-angles.

4.5.2 Convolutional Layers look at a local subset of points, applying (and modifying with backpropagation) a filter that can detect and distill local structures such as lines, boundaries and shapes. This is especially useful for OCR, where local patterns are very significant and important to extract, and could be easily lost. The hypothesis is that the filters and information extracted by adding convolutional layers before running dense layers could breach the 90% accuracy. We set the dense layers as the best performing from the previous tests, three hidden layers of 128, 512, and 256 nodes each.

The ways to vary convolutional layers are fundamentally different from dense layers. We can vary the size of the kernel, how large the local subset we focus on is. And we can vary the number of filters we apply to these kernels, which essentially amounts to the number of distinct local features we want our model to pick up.

Table 5: Convolutional Neural Network Accuracy

Model ID	Hiragana	Kanji	Combined
CNN 1	90.4	89.3	88.8
CNN 2	86.1	89.0	87.9
CNN 3	92.2	94.7	90.4
CNN 4	95.5	96.1	94.9
CNN 5	98.2	97.9	96.6
CNN 6	95.9	96.7	94.1

Table 6: Convolutional Neural Network Accuracy

Model ID	Model Params
CNN 1	C(8, (3,3))
CNN 2	C(8, (5,5))
CNN 3	C(64, (3,3)), C(32, (3,3))
CNN 4	C(64, (3,3)), (0.5x), C(32, (3,3))
CNN 5	C(128, (3,3)), (0.5x), C(128, (3,3)), (0.5x),C(128, (3,3))
CNN 6	C(256, (3,3)), (0.5x), C(64, (3,3)) (0.5x), C(64, (3,3))

As can be seen in Table 6, the best setup we achieved was with CNN 5. It was built with three convolutional hidden layers with 128 filters and a 3x3 kernel, downsampling by half (represented by 0.5x) between each convolutional layer. The convolution step was followed by 3 dense hidden layers with 1024, 512, 256 nodes respectively.

5 Experiments/Evaluation

Our primary metric for evaluation was testing accuracy score, and decidedly the thing we sought to maximize. Through training and tuning models, the results of which can be seen above. Figure 5, Figure 6, and Table 3 show the tuning of Support Vector Machines, which yielded promising results, especially for classifying whether a character was Hiragana or Kanji.

Rather than focusing solely on the numbers, we chose to evaluate our models based on how they misclassified certain characters and what they were classified as. Table 7 shows the results of the worst classified Hiragana for each model, along with their associated accuracy score.

Table 7: Most Misclassified Hiragana for All Models

Model	Char	Acc	Char	Acc	Char	Acc
RF	ひ	38%	べ	39%	ぼ	42%
KNN	ひ	33%	ほ	39%	ぼ	42%
SVM	べ	51%	ぼ	60%	ふ	61%
NN	ぐ	43%	ひ	54%	ふ	54%
CNN	さ	68%	づ	71%	に	71%

This is a fascinating result, and it truly enlightens us on how these models perform, as well as the power of convolution in neural networks. Random Forests, K-Nearest Neighbors, Support Vector Machines and Dense Neural Networks all struggled with the same set of characters. は, ひ, ふ, へ, ほ, ば, び, ぶ, べ, ぼ, ぱ, ぴ, ぷ, ぺ, ぽ, the HA characters, are unique among the Hiragana because they have two different accents that can appear for different voicings. は (HA) becomes ば(BA) with two tick-marks, and becomes ぱ(PA) with a small circle. Interestingly enough, the character was seldom misclassified as the similar character with a different voicing. That is, the SVM did not classify べ as ぺ or as へ often. In fact, both of these characters appear outside of the top 10 misclassifications for this character.

This is difficult to fully dissect and understand, but we speculate that part of the reason for this is the manner in which PCA transformed each character. As was mentioned when analyzing Figure 2, most of the significant Eigengana had either positive weight or negative weight in the top right corner where the voicing accent appears. Overweighting this feature was very useful in distinguishing characters like だ and た, where the accent is the only distinguishing feature, but this then caused the models to struggle when multiple features could appear in tat corner. This also influenced the Dense Neural Network, which could not capture the details in the differences between the accents.

Convolution, however, suffered no problems from the HA characters, and in fact the worst performing characters changed each time the model was run. This gives immense support to the power of CNN's ability to capture and distinguish finer details within the character image, and proves that convolution was the step needed to push densely connected neural networks past the 90% accuracy rate barrier.

Table 8 shows a similar breakdown by the most misclassified Kanji characters. While this is equally interesting to dig into, it is not the smoking gun that Hiragana was in making bare the deficiencies of non-convolutional models for OCR. One interesting result is 者 being unusually difficult to classify for all models. No other Kanji appears in all of the misclassifications, but the other characters it was classified as does not remain consistent across the models. We do not have a good explanation of why this is the case.

Table 8: Most Misclassified Kanji for All Models

Model	Char	Acc	Char	Acc	Char	Acc
RF	曜	26%	者	27%	発	51%
KNN	者	29%	曜	42%	全	47%
SVM	者	51%	実	60%	間	61%
NN	曜	32%	者	51%	全	58%
CNN	者	51%	人	68%	八	69%

6 Final Results

We have two options in regards to best classifying characters. We can either first try to classify a character as Hiragana or Kanji. And then re-run that character across the best model for Hiragana or Kanji. Or we can use the model that best classifies our combined data set.

For the first option, the best model for classifying Hiragana vs Kanji was SVM with a gamma of 0.001 and C of 10000. SVM gives a 98% accuracy in determining Hiragana vs Kanji. Then the best model for Hiragana and the best model for Kanji was the neural network with 98.2% and 97.9% accuracies respectively. For both these models the best configuration for the neural networks were Conv(128 (3,3)), 2x Downsample, Conv(128 (3,3)), 2x Downsample, Conv(128 (3,3)), Flatten, Dense(1024,512,256). When running SVM first and then neural network second, we had a net accuracy rate of 96.06%. In this case, we ran the SVM to classify a character as either Hiragana or Kanji. If the character was Hiragana, we then ran the character through the Hiragana trained neural network. If the character was Kanji we ran the model through the Kanji trained neural network.

However, running a character through a neural network trained on a combined Hiragana + Kanji data set, we had an accuracy score of 96.6%. This is slightly higher than the SVM + neural network approach. Therefore, in order to best classify Japanese characters, a neural network trained on the entire data set is the most accurate approach.

There were two of us working on this project: Bharathrham and Mohnish. Bharathrham is fluent in Japanese and led the effort regarding analyzing Eigengana and determining the likely reasons for character miss-classification. The data cleaning and denoising was split amongst the two of us. In terms of model building, Bharathrham built the neural network models. Mohnish built the Random Forest, SVM, and KNN models. In the end, we both analyzed the results of each model together.

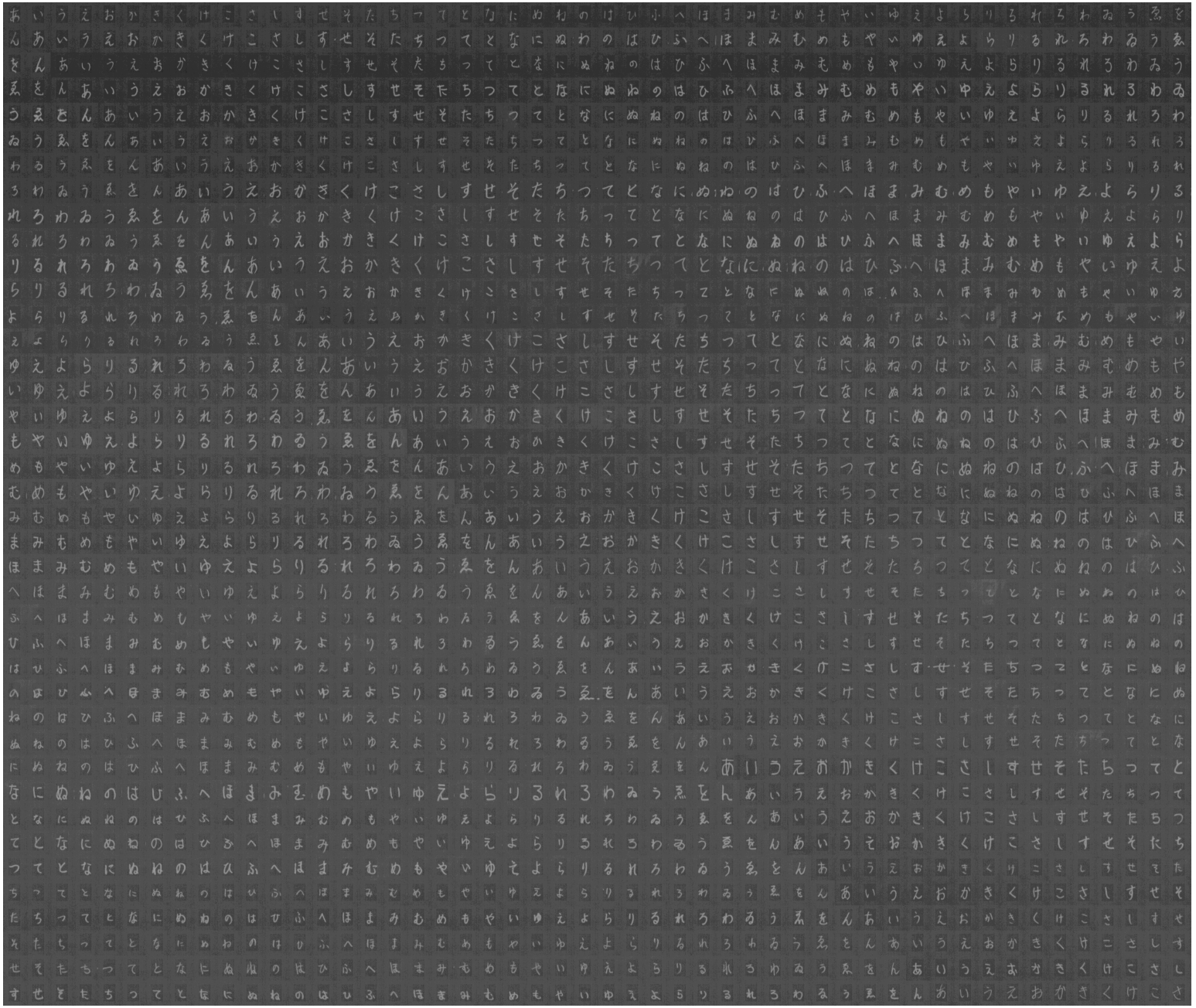

Figure 9: Starting image example. Noisy, needing to be separated, denoised and rescaled